УДК 004.04

ПРИМЕНЕНИЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ ПРИ СОЗДАНИИ РАСПРЕДЕЛЕННЫХ ВЕБ-ПРИЛОЖЕНИЙ

С.Д. ПАШКЕВИЧ
(Представлено: канд. техн. наук, доц. А.Ф. ОСЬКИН)

Рассматриваются основные виды веб-приложений. Предлагается подход к проектированию сложных веб-приложений, основанный на использовании микросервисной архитектуры. Описываются функциональные возможности такого рода приложений, а также возможные средства взаимодействия между микросервисами.

Введение. Развитие интернет-технологий, сервисов, а также интернет-услуг, ставит перед разработчиками и веб-приложениями новые, все более сложные задачи. Возрастающие требования к веб-приложениям, необходимость интеграции со сторонними сервисами, обеспечение необходимой скорости доступа к информации, все эти критерии обуславливают создание сложных приложений, которые будут соответствовать современным стандартам.

Существует два основных вида веб-приложений: монолитные и приложения с микросервисной архитектурой.

Монолитное приложение – это приложение, построенное, как единое целое. Такие приложения часто состоят из трех основных частей: пользовательский интерфейс, база данных и сервер. Серверная часть должна быть способна обрабатывать HTTP запросы в соответствии с базовой логикой, а также возвращать результат выполнения, который может быть использован другими приложениями для дальнейшего отображения конечному пользователю. Все операции, относящиеся к обработке данных, выполняются в единственном процессе, то есть запуск самой простой операции требует инициализации всего приложения, что делает большие монолитные программы ресурсозатратными. Кроме этого, если приложение существует уже достаточно долгое время, в процессе разработки и внедрения новых функциональных возможностей, оно увеличивается до огромных размеров, что в свою очередь требует затрачивать больше времени на разработку. Масштабировать приложение горизонтально приходится целиком, путем его запуска на нескольких физических серверах, чтобы иметь возможность распределять нагрузку.

Эти неудобства привели к архитектурному стилю микросервисов: построению приложений в виде набора сервисов. В дополнение к возможности независимого развертывания и масштабирования каждый сервис также получает четкую физическую границу, которая позволяет разным сервисам быть написанными на разных языках программирования. Они также могут разрабатываться разными командами.

Архитектурный стиль микросервисов – это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и взаимодействует с остальными используя легковесные механизмы. Эти сервисы построены вокруг бизнес-потребностей и инициализируются независимо, с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных [1].

Одним из основных преимуществ микросервисов является возможность изменения отдельно взятого микросервиса, не меняя при этом связанных с ним компонентов.

Таким образом, следует выделить основные особенности:

- упрощается процесс масштабирования разрабатываемого приложения;
- каждый микросервис не зависит от других микросервисов;
- микросервисы могут быть взаимозаменяемы.

Однако, не смотря на все преимущества, существуют некоторые недостатки: требуется учитывать тот факт, что любой из микросервисов может выйти из строя, кроме этого, необходимо предусмотреть реализацию поведения микросервисов и их взаимодействие.

Основная часть. Взаимодействие микросервисов в рамках одного приложения для осуществления поставленной задачи следует выполнять, опираясь на потребности пользователей либо бизнес-логику предприятия, для которого разрабатывается приложение.

Таким образом, при проектировании и разработке приложения, следует учитывать основные термины, с которыми приходится сталкиваться пользователям конечного продукта, разрабатывать компоненты системы, которые обязаны соответствовать бизнес логике, строить приложение опираясь на процессы, с которыми сталкивается пользователь в повседневности, в рамках своей рабочей деятельности, а не на данных.

Предметно-ориентированное проектирование (DDD) – это набор принципов и схем, направленных на создание оптимальных систем объектов. Сводится к созданию программных абстракций, которые называются моделями предметных областей. В эти модели входит бизнес-логика, устанавливающая связь между реальными условиями области применения продукта и кодом.

Предметно-ориентированное проектирование не является какой-либо конкретной технологией или методологией. DDD – это набор правил, которые позволяют принимать правильные проектные решения. Данный подход позволяет значительно ускорить процесс проектирования программного обеспечения в незнакомой предметной области [2].

Исходя из этого, можно сделать вывод, что DDD отлично подходит для непростых предметных областей.

Большинство корпоративных приложений со сложной бизнес-логикой и технической реализацией разделяются на уровни. Уровни являются логическим артефактом и не относятся к развертыванию службы. Они нужны, чтобы помочь разработчикам управлять сложными процессами в коде. Различные уровни (например, уровень модели предметной области, уровень представления данных и т. д.) могут иметь различные типы.

Например, сущность может загружаться из базы данных. Затем часть этих сведений, или объединенные данные, включающие данные из других сущностей, могут отправляться в пользовательский интерфейс клиента. Суть в том, что объект содержится в уровне модели предметной области и не должен передаваться в другие области, к которым он не принадлежит, такие как уровень представления данных [3].

Таким образом, используя DDD и микросервисную архитектуру, можно проектировать достаточно сложные и большие корпоративные приложения. Каждый микросервис обязан реализовывать задачи определенной части из области определения, что гарантирует то, что микросервисы будут независимы не только физически, но и логически. Кроме этого, такой подход позволит одновременно работать разным командам в рамках одного приложения, используя разные языки программирования. У разработчиков одного из микросервисов нет необходимости вникать в функциональность другого микросервиса, что также ускоряет разработку.

Следующий важный шаг, который стоит учитывать при проектировании такого рода систем – взаимодействие между ними.

Существуют следующие подходы:

– взаимодействие посредствам HTTP запросов;
– взаимодействия с помощью очереди событий.

Так как приложение распределенное, то есть, некоторые операции будут выполняться различными сервисами, то следует учитывать тот факт, что не все микросервисы будут доступны в момент выполнения этой операции. Таким образом, если один из микросервисов сделает HTTP запрос на обновление данных в другом микросервисе и тот не будет доступен в данный момент времени, то операция будет прервана, что в свою очередь может стать причиной неактуальных данных. Решить эту проблему позволяют специальные службы, которые способны строить очереди из событий, которые происходят в одном приложении, затем другие приложения отслеживают события из этой очереди, в результате чего, при появлении определенного события, выполняют ту или иную операцию. Такая очередь событий должна обладать способностью накапливать события в том случае, если приложение, для которого событие было предназначено, недоступно. RabbitMQ соответствует этим требованиям.

RabbitMQ – это мультипротокольный брокер сообщений, позволяющий организовать отказоустойчивый кластер с полной репликацией данных на несколько узлов, где каждый узел может обслуживать запросы на чтение и запись [4]. То есть основная цель RabbitMQ – принимать и отдавать сообщения.

RabbitMQ позволяет взаимодействовать различным программам при помощи протокола AMQP, является отличным решением для построения SOA (сервис-ориентированной архитектуры) и распределением отложенных ресурсоемких задач [5].

Таким образом, использовать данную службу, при построении распределенного веб-приложения, является хорошим решением. Кроме этого, такой подход позволит отправить выполнение ресурсоемкой операции в фоновой процесс, в следствии чего, у пользователя нет необходимости ожидать завершение выполнения данной операции, так как результаты будут вычислены, не блокируя пользовательский интерфейс.

Следовательно, RabbitMQ станет посредником при взаимодействии микросервисов в рамках распределенного приложения. Каждый микросервис, в случае необходимости, будет иметь возможность сообщить другим микросервисам о том, что завершена та или иная операция, путем публикации сообщения в очередь. После этого, любой микросервис будет иметь возможность отслеживать необходимые сообщения и реагировать на них заданным образом.

На рисунке представлен пример взаимодействия микросервисов.

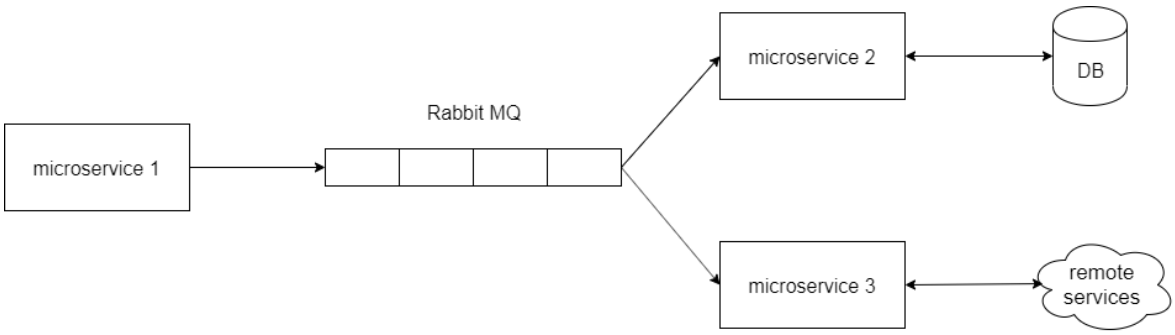

Рисунок. – Пример взаимодействия микросервисов

Заключение. Предложен подход к реализации распределенных веб-приложений. Рассмотрены основные виды веб-приложений, а также средства взаимодействия между микросервисами.

ЛИТЕРАТУРА

1. Habr [Электронный ресурс] / habr.com © 2019. – Режим доступа: https://habr.com/ru/post/249183/. – Дата доступа: 10.09.2019.
2. The Web Land [Электронный ресурс] / thewebland.net © 2019. – Режим доступа: https://thewebland.net/domain-driven-design/. – Дата доступа: 16.09.2019.
3. Microsoft [Электронный ресурс] / Microsoft © 2019. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice. – Дата доступа: 17.09.2019.
4. Habr [Электронный ресурс] / habr.com © 2019. – Режим доступа: https://habr.com/ru/company/tensor/blog/341068/. – Дата доступа: 25.09.2019.
5. Habr [Электронный ресурс] / habr.com © 2019. – Режим доступа: https://habr.com/ru/post/149694/. – Дата доступа: 27.09.2019.